In the United States Patent and Trademark Office

In re the Application of:

Steven Edward Atkin	)	
Serial Number: 09/838,377	)	Group: 2122
Docket Number: AUS920010277US1	)	Examiner: Andre R. Fowlkes
Filed on: 04/19/2001	)	
For: "Bi-Directional Display"	)	

APPEAL BRIEF (Revised)

Real Party in Interest per 37 CFR §41.37(c)(1)(i)

The subject patent application is owned by International Business Machines Corporation of Armonk, NY.

Related Appeals and Interferences per 37 CFR §41.37(c)(1)(ii)

None.

Status of Claims per 37 CFR §41.37(c)(1)(iii)

Claims 1 - 30 were originally filed in the application. Claims 3, 13, and 23 were cancelled by applicant's amendment on January 19, 2005. Claims 1, 2, 4 - 12, 14 - 22, and 24 - 30 were finally rejected in the Office Action dated May 20, 2005. The rejections of Claims 1, 2, 4 - 12, 14 - 22, and 24 - 30 were appealed on August 22, 2005.

Status of Amendments after Final Rejections per 37 CFR §41.37(c)(1)(iv)

Claims 7, 17 and 27 were amended after Final Rejections in order to comply with a suggestion by the Examiner to expand the abbreviation of SML to "Standard Machine Language".

Summary of the Claimed Subject Matter per 37 CFR §41.37(c)(1)(v)

Claims 1, 11, and 21 are independent claims from which all other claims depend. Appellant's invention provides a method and system which converts a logically ordered character stream into a character stream suitable for display by a computer and comprehension by a user for purposes of displaying information in one or more languages, where the languages may have differing orders of character display (e.g. left to right, right to left, etc.). Each logically ordered character stream has a plurality of characters and control codes contained within it.

More specifically, our invention comprises a method and system which (please note that Table 11 is supplied with actual code rather than a drawing or illustration, so references to code line numbers in Table 11 replace customary references to drawing number and item number):

(a) assigns bidirectional attributes to such a logically ordered character stream (pg. 25 lines 10 - 12; pg. 29 lines 11 - 14; Table 11 code lines 1 - 71);

(b) assigns initial level numbers while honoring any directional overrides by explicit processing (pg. 25 lines 13 - 15; pg. 29 lines 14 - 16; Table 11 code lines 121 - 132);

(c) changes attribute types based upon surrounding attribute types through weak and neutral processing (pg. 25 lines 14 - 16; pg. 29 lines 16 - 17; Table 11 code lines 73 - 99);

(d) associates final level numbers to the logical character stream through implicit processing (pg. 25 lines 16 - 17; Table 11 lines 102 - 119); and

(e) reorders the characters within the logical character stream according to the final level numbers such that the reordered characters form a character stream in display order (pg. 25 lines 17 - 18; pg. 29 lines 17 - 18; Table 11 code lines 134 - 158),

(f) wherein facets of layout relating to character reordering and facets related to character stream rendering are handled separately in a functional programming language, and said character stream is handled as sequential runs of integers during said steps of assigning attributes, level numbers, changing, attribute types, associating final level numbers, and reordering characters (pg. 23 lines 19 - 21;

pg. 25 lines 19 - 22; pg. 30 lines 14 - 16; pg. 32 line 20 - pg. 33 line 6; and Table 11 code lines 1 - 158).

Grounds for Rejection For Which Review is Sought per 37 CFR §41.37(c)(1)(vi)

Review by the Board of the rejections of Claims 1, 2, 4 - 12, 14 - 22, and 24 - 30 under 35 U.S.C. §103(a) as being unpatentable over non-patent publication "Implementations of Bidirectional Reordering Algorithms", Florida Tech Technical Report CS-2000-1, by Steven Atkin and Ryan Stansifer (hereinafter "Atkin-Stansifer") in view of non-patent publication "FAQ for comp.lang.functional" by Hutton (hereinafter "Hutton") is requested.

Arguments per 37 CFR §41.37(c)(1)(vii)

Rejections of Claims 1, 11 and 21 under 35 U.S.C. §103(a) over Atkin-Stansifer in view of Hutton

The Atkin-Stansifer publication is the work of the Appellant, Steven Edward Atkin, which was produced during course work while attending Florida Institute of Technology. Dr. Stansifer was appellant's faculty advisor, and as such, acted as reviewer, commenter, and editor for the paper, but otherwise did not contribute to the technical details of the problem recognition and solution as set forth in the paper. Therefore, Dr. Stansifer is a co-author of the cited paper, but is not a co-inventor for the present patent application.

Appellant supplies herewith an affidavit under 37 C.F.R. §1.132 to place into evidence the facts that the cited Atkin-Stansifer paper is the work of the appellant, including evidence of conception of the claimed invention prior to the publication date afforded by Examiner to the Atkin-Stansifer paper in the form of a date-stamped JAVA file and code listing.

Such a showing of fact by appellant is sufficient to overcome a cited reference under 35 U.S.C. 102(a). *In re DeBaun*, 687 F.2d 459, 214 USPQ 933 (C.C.P.A. 1982), and *In re Katz*, 687 F.2d 450, 215 USPQ 14.

The Hutton reference taken alone fails to teach all of the claimed elements, steps, and limitations as set forth in the Office Action of May 20, 2005. For these reasons, Appellant requests reversal of the rejections of claims 1, 11, and 21.

Rejections of Claims 1, 11 and 21 under 35 U.S.C. §103(a) over Atkin-Stansifer in view of Hutton

Claims 2, and 4 - 10 depend from Claim 1; claims 12, and 14 - 20 depend from Claim 11; and claims 22, and 24 - 30 depend from Claim 11. As discussed in the foregoing arguments regarding the independent claims, the Atkin technical report is not available as prior art against appellant's claims.

The Hutton reference taken alone fails to teach all of the claimed elements, steps, and limitations as set forth in the Office Action of May 20, 2005. For these reasons, Appellant requests reversal of the rejections of claims 2, 4 - 10, 12, 14 - 20, 22, and 24 - 30.

Respectfully Submitted,

Robert Frantz

Agent for Appellant(s)
Robert H. Frantz, Reg. No. 42,553
Tel: (405) 812-5613

Franklin Gray Patents, LLC
P.O. Box 23324
Oklahoma City, OK 73127
Tel: 405-812-5613
Fax: 405-440-2465

Claims Appendix

per 37 CFR §41.37(c)(1)(viii)

Clean Form of Amended Claims

Claim 1 (previously presented):

A method of converting a logically ordered character stream into a character stream suitable for display by a computer and comprehension by a user, said logically ordered character stream having a plurality of characters and control codes contained within it, said method comprising:

assigning bidirectional attributes to a logical character stream;

assigning initial level numbers and honoring any directional overrides by explicit processing;

changing attribute types based upon surrounding attribute types through weak and neutral processing;

associating final level numbers to the logical character stream through implicit processing; and

reordering said characters within said logical character stream according to said final level numbers such that said reordered characters form a character stream in display order wherein facets of layout relating to character reordering and facets related to character stream rendering are handled separately in a functional programming language, and said character stream is handled as sequential runs of integers during said steps of assigning attributes, level numbers, changing, attribute types, associating final level numbers, and reordering characters.

Claim 2 (original):

The method as set forth in Claim 1 wherein said step of assigning bidirectional attributes further comprises obtaining said bidirectional attributes from a character database.

Claim 3 (canceled).

Claim 4 (original):

The method as set forth in Claim 1 wherein said step of changing attribute types based upon surrounding attribute types through weak and neutral processing in a functional programming language comprises providing blocks of functional programming language indexed by name weak type processing, neutral type processing, and implicit level processing such that said method may be readily used as a reference.

Claim 5 (original):

The method as set forth in Claim 1 wherein one or more steps are provided in Haskell functional language.

Claim 6 (original):

The method as set forth in Claim 1 wherein one or more steps are provided in Erlang functional language.

Claim 7 (previously presented):

The method as set forth in Claim 1 wherein one or more steps are provided in Standard Machine Language ("SML") functional language.

Claim 8 (original):

The method as set forth in Claim 1 wherein one or more steps are provided in Miranda functional language.

Claim 9 (original):

The method as set forth in Claim 1 wherein one or more steps are provided in Lisp functional language.

Claim 10 (original):

The method as set forth in Claim 1 wherein one or more steps are provided in Scheme functional language.

Claim 11 (previously presented):

A computer readable medium encoded with software causing a computer to perform the following actions:

receiving a logically ordered character stream;

assigning bidirectional attributes to the logical character stream;

assigning initial level numbers and honoring any directional overrides by explicit processing;

changing attribute types based upon surrounding attribute types through weak and neutral processing;

associating final level numbers to the logical character stream through implicit processing; and

reordering said characters within said logical character stream according to said final level numbers such that said reordered characters form a character stream in display order, wherein facets of layout relating to character reordering and facets related to character stream rendering are handled separately in a functional programming language, and said character stream is handled as sequential runs of integers during said steps of assigning attributes, level numbers, changing, attribute types, associating final level numbers, and reordering characters.

Claim 12 (original):

The computer readable medium as set forth in Claim 11 wherein said software for performing said assignment of bidirectional attributes further comprises software for obtaining said bidirectional attributes from a character database.

Claim 13 (canceled):

Claim 14 (original):

The computer readable medium as set forth in Claim 11 wherein said software for performing the action of changing attribute types based upon surrounding attribute types through weak and neutral processing in a functional programming language comprises software organized into blocks of functional programming language indexed by name weak type processing, neutral type processing, and implicit level processing such that said method may be readily used as a reference.

Claim 15 (original):

The computer readable medium as set forth in Claim 11 wherein said software is Haskell functional language.

Claim 16 (original):

The computer readable medium as set forth in Claim 11 wherein said software is Erlang functional language.

Claim 17 (previously presented):

The computer readable medium as set forth in Claim 11 wherein said software is Standard Machine Language ("SML") functional language.

Claim 18 (original):

The computer readable medium as set forth in Claim 11 wherein said software is Miranda functional language.

Claim 19 (original):

The computer readable medium as set forth in Claim 11 wherein said software is Lisp functional language.

Claim 20 (original):

The computer readable medium as set forth in Claim 11 wherein said software is Scheme functional language.

Claim 21 (previously presented):

A text code conversion system for converting logically ordered text streams and displaying said text streams in a display order, said system comprising:

a character stream receiver for receiving a logically ordered character stream;

a bidirectional attribute assignor realized for assigning bidirectional attributes to a received logical character stream;

an initial level assignor for assigning initial level numbers and for honoring any directional overrides by explicit processing;

an attribute type changer realized for changing attribute types based upon surrounding attribute types through weak and neutral;

a final level assignor realized for associating final level numbers to the logical character stream through implicit processing; and

a character resequencer realized for reordering said characters within said logical character stream according to said final level numbers such that said reordered characters form a character stream in display order, wherein facets of layout relating to character reordering and facets related to character stream rendering are handled separately, and said logically ordered character stream is handled as sequential runs of integers by said character stream receiver, said attribute assignor, said initial level assoignor, said type changer, said final level assignor, and said character resequencer, each of which are realized in a functional programming language.

Claim 22 (original):

The text code conversion system as set forth in Claim 21 wherein said bidirectional attributes assignor is adapted to obtain said bidirectional attributes from a character database.

Claim 23 (canceled).

Claim 24 (original):

The text code conversion system as set forth in Claim 21 wherein said attribute type changer attribute type changer comprises blocks of functional programming language indexed by name weak type processing, neutral type processing, and implicit level processing such that said method may be readily used as a reference.

Claim 25 (original):

The text code conversion system as set forth in Claim 21 wherein said bidirectional attribute assignor, initial level assignor, attribute type changer, final level assignor, and character resequencer comprise Haskell functional language.

Claim 26 (original):

The text code conversion system as set forth in Claim 21 wherein said bidirectional attribute assignor, initial level assignor, attribute type changer, final level assignor, and character resequencer comprise Erlang functional language.

Claim 27 (previously presented):

The text code conversion system as set forth in Claim 21 wherein said bidirectional attribute assignor, initial level assignor, attribute type changer, final level assignor, and character resequencer comprise Standard Machine Language ("SML") functional language.

Claim 28 (original):

The text code conversion system as set forth in Claim 21 wherein said bidirectional attribute assignor, initial level assignor, attribute type changer, final level assignor, and character resequencer comprise Miranda functional language.

Claim 29 (original):

The text code conversion system as set forth in Claim 21 wherein said bidirectional attribute assignor, initial level assignor, attribute type changer, final level assignor, and character resequencer comprise Lisp functional language.

Claim 30 (original):

The text code conversion system as set forth in Claim 21 wherein said bidirectional attribute assignor, initial level assignor, attribute type changer, final level assignor, and character resequencer comprise Scheme functional language.

Evidence Appendix

per 37 CFR §41.37(c)(1)(ix)

Please see attached affidavit under 37 CFR §1.132, totaling 10 pages.

Related Proceedings Appendix

per 37 CFR §41.37(c)(1)(x)

No decisions have been rendered by a court or the Board in the related proceedings as identified under 37 CFR §41.37(c)(1)(ii).

In the United States Patent and Trademark Office

In re the Application of:

Steven Edward Atkin	)	
Serial Number: 09/838,377	)	Group: 2122
Docket Number: AUS920010277US1	)	Examiner: Andre R. Fowlkes
Filed on: 04/19/2001	)	
For: "Bi-Directional Display"	)	

AFFIDAVIT UNDER 37 CFR §1.132

I, STEVEN EDWARD ATKIN, Ph.D., declare as follows:

1. I am the inventor and applicant of the invention entitled "Bi-Directional Display", disclosed and claimed in U.S. Patent Application Serial No. 09/838,377 filed on April 19, 2001.

2. The invention described and claimed in said application was conceived by me prior to October 4, 2000, as evidenced by the following facts which are of my own knowledge.

(a) The paper cited by the Examiner of said patent application, entitled "Implementations of Bidirectional Reordering Algorithms", Florida Tech Technical Report CS-2000-1, by Steven Atkin and Ryan Stansifer was authored by me. During my attendance at Florida Institute of Technology, Dr. Stansifer was my faculty advisor, and as such, acted in an advisory, reviewer and editor role only. This Technical Report was written during the development of my invention as claimed in said Patent Application. The Technical Report number was assigned upon my request on October 4, 2000, in advance of completion of the actual report, as the policy of the Technical Report Library was to assign report numbers upon submission of a title, author name, and abstract.

(b) I conceived of my invention prior to October 4, 2000, as evidenced by Appendix A to this affidavit, which includes several code files dated in June of 2000. These early prototypes of my invention were developed in order to test and prove the validity of the concept, some of the results of which were provided in the disclosure of the Patent Application.

(c) Following completion of verification testing and prototype code development, a disclosure was submitted to inside patent counsel for my employer, International Business Machines, on January 26, 2001. Shortly thereafter, I was notified that the disclosure had been selected for patenting, and I was contacted by outside patent counsel Robert Frantz. This effort resulted in the filing of the Patent Application on April 19, 2001.

3. As such, I am the sole inventor of the invention described and claimed in the Patent Application. Dr. Stansifer provided advisory input only to the cited Technical Report, and thus is only a co-author, but not a co-inventor. Further, I conceived of and prototyped the invention disclosed and claimed in the Patent Application prior to October 4, 2000.

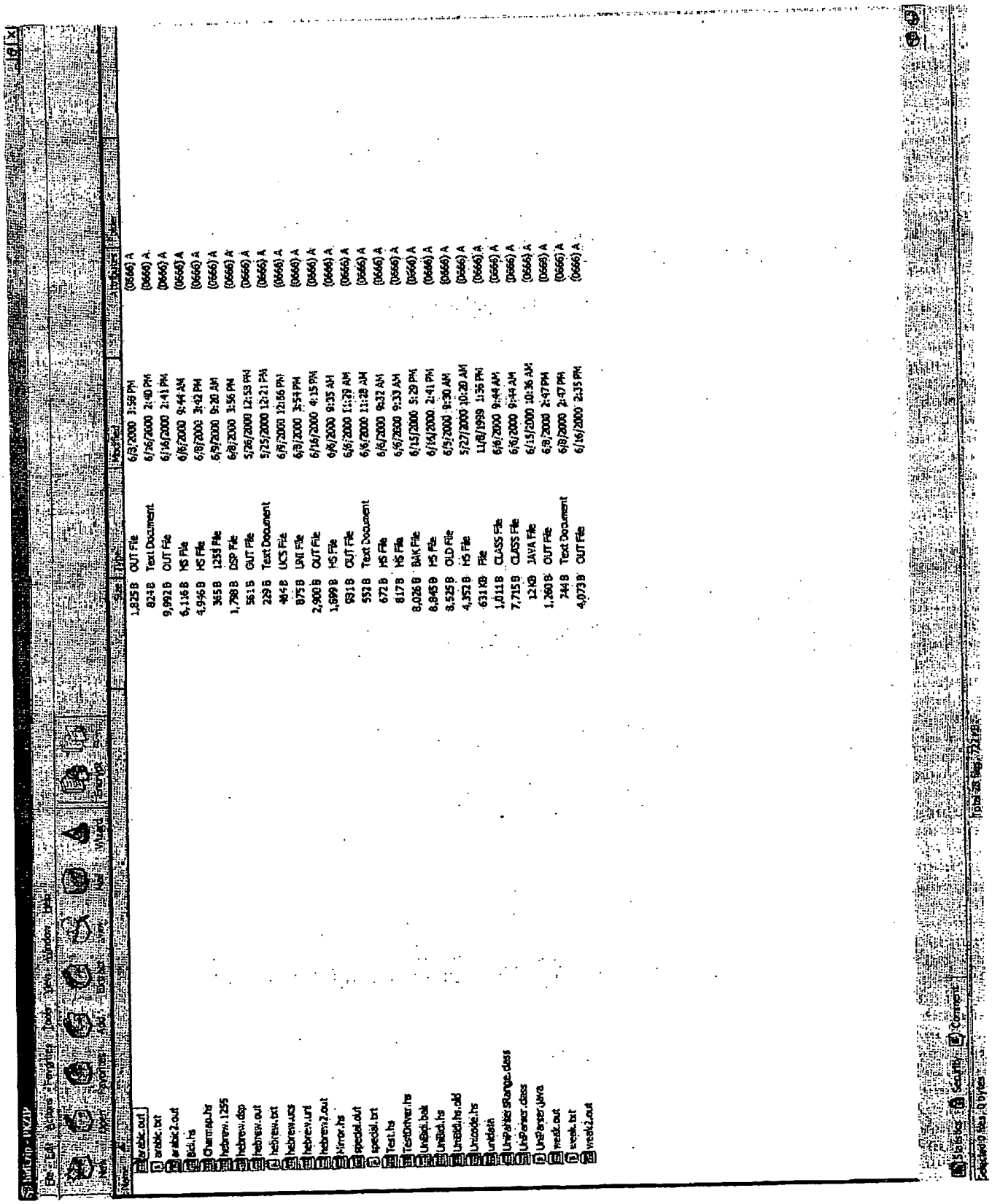
Steven Edward Atkin

Sworn and subscribed to before me this _____ day of _____, 2005.

_____

Notary Public

My Commission Expires _____



PRINT OUT OF UniParser.java, file dated 6/15/2000

```
/*  UniParser.java
 *  Steven Atkin 4/26/2000
 *  This class provides methods for parsing the Unicode data table published
 *  by the Unicode consortium www.unicode.org.
 */

import java.io.*;
import java.lang.*;
import java.text.*;
import java.util.*;
import javax.swing.*;

/**
 * UniParser parses the Unicode data table for a specific attribute
 * and then generates a Haskell 98 (Hugs 98) module for accessing the
 * attribute information.
 */
public class UniParser {
    public class Range{
        int start, end;

        public int getStart() {
           return start;
        }
        public int getEnd() {
           return end;
        }
        public void setStart(int begin) {
            start = begin;
        }
        public void setEnd(int last) {
            end = last;
        }
        public String toString() {
            return ("(0x" +
                    Integer.toHexString(start) +
                    ",0x" +
                    Integer.toHexString(end) +
                    ")");
        }
    }

    int unicodeField = 0, attributeField = 0;
    BufferedReader dataIn;
    BufferedWriter dataOut;
    String delim = ";";
    String moduleName, typeName, functionName;
    String attributes[];
    String defaultAttrib;
    Vector tables[];

    /**
      * Constructor - must specify input data file and
      * output module name.
      */
    public UniParser(String in, String out) {
        try {
            dataIn = new BufferedReader (new FileReader(in));
```

```
            FileOutputStream fileOut = new FileOutputStream (out);
            OutputStreamWriter str = new OutputStreamWriter (fileOut,
"windows-1252");
            dataOut = new BufferedWriter (str);
        }
        catch (UnsupportedEncodingException e1) {
            System.out.println(e1.toString());
            return;
        }
        catch (IOException e) {
            System.out.println(e.toString());
            return;
        }
    }

    /**
      * Parse the individual line for specified token
      */
    private String getToken(String data, int n) {
        StringTokenizer parser;
        String token = "";
        parser = new StringTokenizer(data, delim);
        while(parser.hasMoreTokens() && n > 0) {
            token = parser.nextToken();
            --n;
        }
        return token;
    }

    /**
      * Determine the Unicode codepoint value
      */
    private String getCodePoint(String data) {
        return getToken(data, unicodeField);
    }

    /**
      * Get the attribute field
      */
    private String getAttribute(String data) {
        return getToken(data, attributeField);
    }

    private void insert(Range r, String bi) {
        /* Add the codepoint to the corresponding vector */
            for(int i = 0; i < attributes.length; ++ i) {
                if(bi.equals(attributes[i])) {
                    tables[i].addElement(r);
                    break;
                }
            }
    }

    /**
      * Parse the entire data file for tokens
      */
    public void parse() {
        String in, cp, bidiattr, lastattr = "";
        int unival, lastunival = 0;
        Range range = new Range();
```

```
        StringTokenizer parser;
        try {
            /* Read one line at a time */
            while((in = dataIn.readLine()) != null) {
                cp = getCodePoint(in);
                unival = Integer.parseInt(cp, 16);
                /* Get the attribute value */
                bidiattr = getAttribute(in).toLowerCase();

                if(unival == 0) {
                        lastattr = bidiattr;
                        range = new Range();
                        range.setStart(0);
                }

                if(unival == 0xfffd) {
                        range.setEnd(unival);
                        insert(range, lastattr);
                }

                if(!bidiattr.equals(lastattr)) {
                        range.setEnd(lastunival);
                        insert(range, lastattr);
                        range = new Range();
                        range.setStart(unival);
                        lastattr = bidiattr;
                }
                lastunival = unival;
            }
            dataIn.close();
        }
        catch (IOException e) {
            System.out.println("error parsing data file");
            return;
        }
    }

    /**
      * Turn the vector's contents into a string
      */
    private String toString(Vector v) {
        int max = v.size() - 1;
        StringBuffer buf = new StringBuffer();
        Enumeration e = v.elements();
        buf.append("[");

        for (int i = 0 ; i <= max ; i++) {
           String s = e.nextElement().toString();
           buf.append(s);
           if (i < max)
           buf.append(", ");
            /* Write four per line */
            if ((i+1)%4 == 0 && (i < max))
               buf.append("\n    ");
        }
        buf.append("]");
        return buf.toString();
    }

    /**
```

```
     * Write the Haskell module header
     */
    private void writeModuleHeader() {
        Date date = new Date();
        DateFormat df = DateFormat.getDateTimeInstance(
                            DateFormat.FULL, DateFormat.FULL);
        try {
            dataOut.write("-- This file is computer generated do not modify");
            dataOut.newLine();
            dataOut.write("-- This is a Haskell 98 script file for Hugs98");
            dataOut.newLine();
            dataOut.write("-- " + moduleName + ".hs created on " + df.format
(date));
            dataOut.write(" by java UniParser");
            dataOut.newLine();
            dataOut.write("-- This module provides functions for obtaining " +
                          "Unicode character attributes");
            dataOut.newLine();
            dataOut.write("module " + moduleName + " (");
            dataOut.newLine();
            dataOut.write("    " + functionName + ", ");
            dataOut.newLine();
            dataOut.write("    " + typeName + ",");
            dataOut.newLine();


            for(int i = 0; i < attributes.length; ++i) {
                dataOut.write("    " + attributes[i].toUpperCase());
                if(i != attributes.length - 1)
                    dataOut.write(",");
                dataOut.newLine();
            }
            /* Write the default type */
            if(defaultAttrib.length() != 0)
                dataOut.write("    , " + defaultAttrib.toUpperCase());
            dataOut.newLine();

            dataOut.write(")" + " where");
            dataOut.newLine();
        }
        catch (IOException e) {
            System.out.println("error writing module header");
            return;
        }
    }

    /**
     * Create the attribute type for the haskell module
     */
    private void writeModuleTypes() {
        try{
            dataOut.write("data " + typeName + " = ");
            dataOut.newLine();

            /* Write each attribute as a type constructor
             * with no arguments */

            for(int i = 0; i < attributes.length; ++i) {
                dataOut.write("    " + attributes[i].toUpperCase());
                if(i != attributes.length - 1)
```

```
                dataOut.write(" |");
            dataOut.newLine();
        }
        /* Write the default type */
        if(defaultAttrib.length() != 0)
            dataOut.write("    | " + defaultAttrib.toUpperCase());
        dataOut.newLine();
        dataOut.write("    deriving(Eq, Show)");
        dataOut.newLine();
    }
    catch (IOException e) {
        System.out.println("error writing module types");
        return;
    }
}

/**
  * Write the test function for determing the attribute
  * of a Unicode character.
  */
private void writeModuleAccessFunc() {
    try {
        dataOut.newLine();
        dataOut.write("-- (Start range, End range)");
        dataOut.newLine();
        dataOut.write("member :: Integer -> [(Integer, Integer)] ->
Bool");
        dataOut.newLine();
        dataOut.write("member a [] = False");
        dataOut.newLine();
        dataOut.write("member a ((x,y):xs)");
        dataOut.newLine();
        dataOut.write("    | a < x");
        dataOut.newLine();
        dataOut.write("      = False");
        dataOut.newLine();
        dataOut.write("    | a <= y");
        dataOut.newLine();
        dataOut.write("      = True");
        dataOut.newLine();
        dataOut.write("    | otherwise");
        dataOut.newLine();
        dataOut.write("      = member a xs");
        dataOut.newLine();
        dataOut.newLine();
        dataOut.write(functionName + " :: Int -> " + typeName);
        dataOut.newLine();
        dataOut.write(functionName + " x");
        dataOut.newLine();
        /* Check for all attributes */
        for(int i = 0; i < attributes.length; ++i) {
            dataOut.write("    | member (toInteger x) " + attributes[i]);
            dataOut.newLine();
            dataOut.write("      = " + attributes[i].toUpperCase());
            dataOut.newLine();
        }
        /* Check for default attribute */
        if (defaultAttrib.length() != 0) {
            dataOut.write("    | otherwise");
            dataOut.newLine();
```

```
                dataOut.write("      = " + defaultAttrib.toUpperCase());
                dataOut.newLine();
            }
        }
        catch (IOException e) {
            System.out.println("error writing module accessor function");
            return;
        }
    }

    /**
     * Write each vector out as a table.
     */
    private void writeModuleTables() {
        try {
            for(int i = 0; i < attributes.length; ++i) {
                dataOut.write(attributes[i] + " = " + toString(tables[i]));
                dataOut.newLine();
            }
        }
        catch (IOException e) {
            System.out.println("error writing module tables");
        }
    }

    /* Set the name of the Haskell module */
    public void setModuleName(String name) {
        moduleName = name;
    }

    /* Set the data type name for the Haskell module */
    public void setTypeName(String name) {
        typeName = name;
    }

    /* Set the name of the test function for the Haskell module */
    public void setAccessFuncName(String name) {
        functionName = name;
    }

    /* Set which column the Unicode field is in */
    public void setUnicodeField(int num) {
        unicodeField = num;
    }

    /* Set which field to parse */
    public void setParseField(int num) {
        attributeField = num;
    }

    /* Specify the list of attributes to parse for */
    public void setAttributeArray(String[] attribs) {
        attributes = attribs;
        tables = new Vector[attributes.length];
        for(int i = 0; i < tables.length ; ++ i) {
            tables[i] = new Vector();
        }
    }

    /* Specify the default attribute */
```

```
    public void setDefaultAttribute(String name) {
        defaultAttrib = name;
    }

    /**
      * Write the Haskell module file.
      */
    public void writeModule() {
        writeModuleHeader();
        writeModuleTypes();
        writeModuleTables();
        writeModuleAccessFunc();
        try {
            dataOut.close();
        }
        catch (IOException e) {
            System.out.println("error closing module file");
            return;
        }
    }

    /* Driver */
    public static void main(String[] args) {
        String attributes[] = {"r", "al", "en", "es", "et", "an", "cs",
                               "nsm", "bn", "b", "s", "ws", "on", "lre",
                               "rle", "pdf", "lro", "rlo", "n"};
        UniParser p = new UniParser("unidata", "Bidi.hs");
        p.setModuleName("Bidi");
        p.setTypeName("Bidi");
        p.setAccessFuncName("getBidiAttr");
        p.setUnicodeField(1);
        p.setParseField(5);
        p.setAttributeArray(attributes);
        p.setDefaultAttribute("l");
        p.parse();
        p.writeModule();
    }
}
```